Optimization of a Symmetric Block Lanczos Basis Generation Process

Osni A. Marques

CERFACS Report TR/PA/93/52

Optimization of a Symmetric Block Lanczos Basis Generation Process

Osni A. Marques[†]

November 1993

Abstract

This report describes a set of experiments intended for the optimization of a block Lanczos based algorithm, for the solution of large, sparse, symmetric generalized eigenproblems. The algorithm examined is implemented with a shift-invert operator, using a dynamic shift to slice the eigenvalue spectrum. However, such a shift strategy is application dependent, since it is related to the number of solutions required, the computational costs, the convergence rates and the approximate eigenvalue distribution computed. Our objective is to examine the vector generation process, trying to speed it up by using appropriate basic linear algebra kernels. The fundamentals of the algorithm and the most important points in terms of computational effort are first revised. Then, their performances are examined by a simulation of different problem dimensions and block sizes. The experiments are performed on high performance workstations and shared memory multiprocessors. Finally, some current applications are discussed, as well as subjects for future work.

[†]CERFACS, Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique, 42 av. G. Coriolis, 31057 Toulouse Cedex, France, e-mail: marques@cerfacs.fr

1 Introduction

The determination of solutions of the equation

$$Ax = \lambda Bx \tag{1}$$

where A and B are $n \times n$ matrices, x is a non null vector and λ is a scalar, has long been an important computation. In many applications, the above *generalized eigenproblem* has a companion differential equation describing a physical phenomenon, so that the n possible solutions (λ, x), *eigenvalues* λ and *eigenvectors* x, are associated with fundamental characteristics of such phenomena. In some applications from chemistry, for instance, they represent basic configurations of molecules (hinge-bending motions); in structural engineering, dynamic properties of a given model (natural vibration frequencies and mode shapes); and in nuclear power plants, neutron fluxes (if the dominant eigenvalue is greater than one the behaviour of the plant is super-critical). See [4] and [29] for different cases and meanings. Depending on the level of the discretization of a continuous problem or the precision required for the results, A and B can reach dimensions of many thousands. Although in practical analyses only a few *eigenpairs* (λ, x) are taken into account, either in the extremities of the spectrum (lower or upper) or in an interval (ξ_1, ξ_2), their evaluation is usually a time consuming task. Therefore, the development of new eigensolvers, or the improvement of existing ones, has been the subject of continuous research.

Nowadays, Krylov subspaces based methods are widely used for treating eigenproblems associated with large sparse matrices, since they can be shown to perform better than vector iteration (inverse or direct), transformation methods (Jacobi, Householder or Givens) or determinant search (see [14] and [25] for an explanation of these techniques). It should be noted, however, that if an estimate for some eigenvectors is available, a subspace iteration procedure may benefit from it. This situation occurs when eigenproblems are repeatedly solved for slightly different matrices, as in a finite element modelling phase in structural engineering.

The Lanczos algorithm is an efficient tool for finding a few solutions of large, sparse, symmetric eigenproblems. It can be presented in different ways, as can be seen in [14], [22] and [25], but the governing idea is the generation of an orthonormal basis for a Krylov subspace defined from a starting vector q_1 and a matrix A, i.e.,

$$\mathcal{K}(A, q_1, j) = span(q_1, Aq_1, ...A^{j-1}q_1). \tag{2}$$

The projection of the original problem into this basis leads to a smaller one, involving a symmetric tridiagonal matrix, from which eigenvectors are recovered through a Rayleigh-Ritz procedure. If the generalized eigenproblem (1) is considered, one could define $A_d y = \lambda y$, generating a basis with A_d, where $A_d = L^{-1}AL^{-T}$ and $x = L^{-T}y$, providing B can be factorized as LL^T. Another possibility consists in defining $A_t x = \theta x$, generating a basis with A_t, where $A_t = A^{-1}B$ and $\theta = 1/\lambda$, providing A is invertible. In this case A_t is nonsymmetric, but self-adjoint with respect to B, and its eigenvalues are real (this operator also appears in the subspace iteration context) [25]. It is usually assumed (and observed) that the Lanczos algorithm finds solutions at both ends of the spectrum. However, its convergence pattern is strongly related to the eigenvalue distribution. In addition, experience shows that convergence is faster to the small eigenvalues (i.e., their reciprocals) when the

basis is generated with the operator A_t. On the other hand, a spectral transformation can be applied, in which A is replaced by $A_\sigma = A - \sigma B$, for a real σ, and the algorithm will converge to solutions around σ, in a modified coordinate system. As originally proposed by Ericsson and Ruhe [9], the spectral transformation was implemented using the operator A_d, but Nour-Omid *et al.* [23] showed that is preferable to use A_t. Actually, both operators are not explicitly formed, but B is often semidefinite and the eigenvectors of A_t are directly related to those of the original problem. Still, with an adequate σ, clustered eigenvalues are mapped into A_t in a different way (i.e., $\lambda = \sigma + 1/\theta$), leading to better convergence rates.

The Lanczos algorithm can be also applied to the solution of nonsymmetric eigenproblems through the generation of two sequences of orthonormal vectors (or a biorthogonal procedure, for A and A^T), which project the original problem into an unsymmetric tridiagonal matrix. Rajakumar and Rogers [28], for instance, applied the technique to the solution of a fluid-structure interaction problem. Moreover, Ye [32] developed a convergence analysis, defining approximation bounds for the computed eigenpairs (also suitable for the symmetric case). However, a zero scalar product of two non null vectors can be obtained by the biorthogonal strategy, halting the basis generation process. Although Freund *et al.* [10, 12, 11] proposed a "look-ahead" scheme for avoiding such a breakdown, the Arnoldi algorithm has been preferred for the solution of nonsymmetric eigenproblems, since it generates only one sequence of vectors and is thus stable. In this case, the projection of the original problem into the basis is represented by an upper Hessenberg matrix. Braconnier [3], and Godet-Thobie [13], for instance, describe an Arnoldi based implementation improved with Tchebyshev polynomial accelerators and an eigenvector deflation method. On the other hand, Sorensen [31] proposes an iterative Arnoldi procedure, which tries to find an invariant subspace through the implicit application of polynomial filters.

Different versions of the symmetric Lanczos have been suggested and a comprehensive one is described by Nour-Omid in [22]. The first experiments with the algorithm on a vector computer were possibly carried out by Parlett *et al.* [26], followed by Jones and Patrick [17, 18], who developed a code for shared memory multiprocessors. Bostic and Fulton [2] adopted a different approach, using a spectral transformation scheme with a different σ on each processor of a parallel computer. Within this approach, different processors may compute the same solutions if their σ are not sufficiently separated, and some expertise has to be used to identify such solutions. All these aforementioned implementations are based on the addition of a new vector into the basis at each step. Another proposed variant is the *s-step* procedure [5, 20, 19], in which s steps of the basic Lanczos are performed at the same time, starting with a subspace defined as $span(q_1, Aq_1, A^2q_1 ... A^{s-1}q_1)$. The orthogonality among the generated vectors is established by factors computed through $3s - 1$ systems of linear equations of order s. Although this variant requires additional operations, it improves data locality and has interesting parallel properties. It should be noted, however, that the *s-step* terminology is also used, in a different context, to refer to an iterative fashion of computing selected eigenvalues through the Lanczos algorithm [14].

Block versions of the Lanczos algorithm have been also proposed [15, 16], aiming at a better performance when there are multiple eigenvalues. This is related to the fact that the basic Lanczos algorithm produces only unreduced tridiagonal matrices, and the eigenvalues of such matrices are distinct. In a block method, a basis is generated from p orthogonal

starting vectors, $Q_1 = [\ q_1^{(1)}\ q_2^{(1)}\ ...\ q_p^{(1)}\ ]$, for a block Krylov subspace defined as

$$\mathcal{K}(A, Q_1, j) = span(Q_1, AQ_1, ...A^{j-1}Q_1), \tag{3}$$

so that the projection of the original problem into the basis is given by a block symmetric tridiagonal matrix. Still, A can be replaced by either A_d or A_t, and it can be shown that the overall convergence pattern improves with increasing p [14]. Nevertheless, it is often suggested that the block size should be specified as following [24]:

$p \leq m$, where m is the number of required eigenpairs;

$p \geq 6$, for structural engineering analysis problems with rigid body motions (there is a similar situation in chemistry applications);

$p \geq k$, where k is the largest multiplicity of any sought eigenvalue (which is normally unknown in advance).

In addition, the application of A on p vectors should compensate for the overhead of retrieving it, since A can be stored in a way that profits from its pattern (sparse storage, for instance), or even stored in a slow access memory. The computational cost for a block-step is also more expensive than the one for a single-step, and some ratio between such costs could be considered for defining a maximum p or a default, when no information about the problem to be solved is available. Therefore, an optimal block size depends on many factors, possibly including the computer architecture, and the determination of a p that would lead to consistent performance on all aplications is not feasible.

This report describes some experiments intended for the improvement of a block Lanczos based algorithm, for the solution of large, sparse, symmetric generalized eigenproblems. The technique considered is implemented with a shift-invert operator, with a dynamic shift selection, following ideas proposed by Grimes *et al.* [16]. A dynamic shift is useful when the eigenvalue distribution is difficult (clustered, for instance), many solutions are required, or the continuation of a given run is costly when compared with the generation of a new A_σ. In this case, it would be preferable to restart, thus modifying the convergence rate. Then, the eigenproblem might be solved by pieces, based on the spectrum that the Lanczos algorithm approximates as the basis sizes increase. However, since the shift strategy is application dependent, the approach adopted in this work focuses on good implementations of the vector generation process. The application of A on p vectors is not discussed here, since it is influenced by the pattern of the matrices and in many cases dominates the computational costs. The objective is to examine how efficiently the vectors can be generated for different values of p, using different levels of BLAS kernels, which are currently available in almost all computer scientific libraries. In the following sections, the block Lanczos idea and related shortcomings are briefly revised, the points for possible enhancements are identified, and some simulations are locally performed with different problem dimensions and block sizes. The target machines are high performance workstations SUN 10/41 and IBM Risc 6000/950, and shared memory multiprocessors Convex C220, Alliant FX/80, CRAY 2 and CRAY C90. Finally, some current applications are discussed, as well as subjects for future work.

Table 1: Block Lanczos Algorithm

---

set $Q_0 = 0$ and $\mathsf{B}_1 = 0$
set $Q_1 \neq 0$ so that $Q_1^T B Q_1 = I$
define σ and factorize $A_\sigma = A - \sigma B = LDL^T$

For j=1,2,...
a) solve $(LDL^T)R_j = BQ_j$
b) $R_j \leftarrow R_j - Q_{j-1}\mathsf{B}_j^T$
c) $\mathsf{A}_j \leftarrow Q_j^T B R_j$
d) $R_j \leftarrow R_j - Q_j \mathsf{A}_j$
e) factorize $R_j : R_j = Q_{j+1}\mathsf{B}_{j+1}$, $Q_{j+1}^T B Q_{j+1} = I$

---

2 Block Lanczos Algorithm Fundamentals

The block Lanczos basis generation strategy, for eigenproblem (1), transformed into $A_t x = \theta x$, is summarized in Table 1. As can be seen in that table, the basis is constructed from a rank p starting block Q_1, possibly defined with random generated columns. Basically, at each step, a simultaneous inverse iteration on the product BQ_j leads to the "residual" R_j, which is orthogonalized against the two previous blocks, Q_{j-1} and Q_j. The factorization of the residual results in the next set of p B-orthogonal vectors, Q_{j+1}. Thus, Q_j and R_j are $n \times p$ matrices, B_j is $p \times p$ upper triangular

$$\mathsf{B}_j = \begin{bmatrix} \beta_{1,1} & \beta_{1,2} & \dots & \beta_{1,p} \\ 0 & \beta_{2,2} & \dots & \beta_{2,p} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 0 & 0 & \dots & \beta_{p,p} \end{bmatrix}, \tag{4}$$

A_j is pxp, defined as

$$\mathsf{A}_j = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \dots & \alpha_{1,p} \\ \alpha_{2,1} & \alpha_{2,2} & \dots & \alpha_{2,p} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha_{p,1} & \alpha_{p,2} & \dots & \alpha_{p,p} \end{bmatrix}, \tag{5}$$

and one can write (since Q_{j-1} and Q_j are orthogonal)

$$R_j = Q_{j+1}\mathsf{B}_{j+1} = A_\sigma^{-1} B Q_j - Q_j \mathsf{A}_j - Q_{j-1}\mathsf{B}_j^T. \tag{6}$$

After j steps the blocks of vectors generated can be arranged as (see [14], [22] and [25] for more details)

$$\mathcal{Q}_j = \left[\begin{array}{cccc} Q_1 & Q_2 & \ldots & Q_j \end{array} \right] \tag{7}$$

satisfying

$$\mathcal{Q}_j^T B \mathcal{Q}_j = I \tag{8}$$

so that

$$\mathcal{Q}_j^T B A_\sigma^{-1} B \mathcal{Q}_j = T_j \tag{9}$$

where

$$T_j = \left[\begin{array}{ccccc} \mathsf{A}_1 & \mathsf{B}_2^T & & & \\ \mathsf{B}_2 & \mathsf{A}_2 & \mathsf{B}_3^T & & \\ & \mathsf{B}_3 & \mathsf{A}_3 & \ddots & \\ & & \ddots & \ddots & \mathsf{B}_j^T \\ & & & \mathsf{B}_j & \mathsf{A}_j \end{array} \right]. \tag{10}$$

Therefore, the projection of the eigenproblem (1) into the basis defined by relation (7) is the symmetric block tridiagonal matrix T_j, with an approximate solution $(\hat{\lambda}_i, \hat{x}_i)$ given by

$$\hat{\lambda}_i = \sigma + \frac{1}{\theta_i} \tag{11}$$

and

$$\hat{x}_i = \mathcal{Q}_j s_i \tag{12}$$

with (θ_i, s_i) satisfying

$$T_j s_i = s_i \theta_i. \tag{13}$$

This reduced eigenproblem can be solved by means of Givens rotations, for example, and $\hat{x}_i^T B \hat{x}_i = 1$ providing $s_i^T s_i = 1$. The residual of $(\hat{\lambda}_i, \hat{x}_i)$ can be estimated *a priori*, through

$$\| A_t \hat{x}_i - \theta_i \hat{x}_i \| = \| \mathcal{Q}_j (T_j - \theta_i s_i) + R_j E_j s_i \| = \| R_j E_j s_i \| = \| \mathsf{B}_{j+1} s_j^{(i)} \| = \beta_j^{(i)} \tag{14}$$

where $\| \, . \, \|$ is an Euclidian norm with respect to B, $E_j = [\, 0 \; 0 \; \ldots \; I \,]$, and $s_j^{(i)}$ stores the bottom p elements of s_i. Then, the monitoring of the backward error $\beta_j^{(i)} / \| A_t \|$ allows the counting of the converged solutions, by comparing with a specified tolerance, and the finalization of a given run.

The practical implementation of a Lanczos eigensolver requires some care and this work gives a general picture of the situation for the block strategy, see [16] and [25] for details (it should be noted that for $p = 1$ the above formulation reduces to the basic Lanczos scheme). To begin with, the maximum allowable j can be reached without the convergence of all desired eigensolutions. Then, a restart can be performed, taking for Q_1 some linear combination of current Ritz vectors $\hat{x}$ with backward errors not satisfying the specified tolerance. A new σ can be possibly used, probably resulting in an indefinite matrix A_σ, whose LDL^T factorization should be performed with some kind of pivoting[‡]. However, in this case, the errors introduced by an ill-conditioned system will have strong eigenvector

[‡]It should be noted that the *inertia* of A_σ, the number of eigenvalues of A_σ less, equal or greater than zero, can be recovered from D.

components, as shown by Parlett [25, pages 63–65]. The situation can be also monitored using information provided by the eigensolver, i.e., the approximate spectrum allows the estimation of the conditioning of A_σ. On the other hand, one should avoid taking σ close to an eigenvalue because the factors L and D might not exist.

Concerning the vector generation process, A_j is symmetric in theory, but can slightly deviate from symmetry due to roundoff errors. This is taken into account in order to assure a strong level of orthogonality between Q_j and R_j, in operations c and d in Table 1. After this orthogonalization, only its upper triangle is preserved, since this simplification does not influence the following computations. Then, at the end of each step, R_j is factorized as $Q_{j+1}\mathsf{B}_{j+1}$ using a modified Gram-Schimdt orthogonalization strategy [1, 6, 21]. This factorization consists basically in the normalization of a given column and its purging from the remaining ones (to the right). However, one sweep on the p columns may be not enough to produce $Q_{j+1}^T B Q_{j+1} = I$, so that the factorization should be performed in an iterative fashion, repeated up to p times if required. Some perturbation can be also introduced at this point, for the columns of R_j are updated, and an additional orthogonalization is usually performed between Q_j and Q_{j+1}. Because of the inverse iteration in phase a, for example, the columns of R_j tend to converge to the eigenvector closest to σ. However, a rank deficient block can be identified by the conditioning of the final B_{j+1}. Finally, a loss of orthogonality among the vectors of the basis $\mathcal{Q}_j$ is generally observed after some steps. It is caused by the introduction of roundoff errors in relation (6). Paige [25] showed that the departure from orthogonality is also related to the convergence of a pair $(\hat{\lambda}_i, \hat{x}_i)$ and, therefore, with the eigenvalue distribution of the associated problem. Once orthogonality is lost, property (8) is no longer satisfied, and redundant copies of eigenpairs emerge. As an immediate option to avoid this, one could apply a full reorthogonalization, i.e., orthogonalize R_j against all $j-1$ previous blocks. However, such a scheme would strongly increase the number of operations at each step. On the other hand, some preventive measures based on potentially dangerous vectors can be used to keep the orthogonality within a certain level [27, 30], namely:

Selective Orthogonalization. The objective is to keep R_j orthogonal to any previously converged $\hat{x}_i$. Then, whenever the backward error satisfies a specified tolerance the corresponding $\hat{x}_i$ is calculated and stored. In the following steps, the residual block R_j is orthogonalized against $\hat{x}_i$ whenever indicated by the norm $\tau_{j+1}^{(i)} = \| \hat{x}_i^T B Q_{j+1} \|$. The tolerance for $\beta_j^{(i)}/\|A_t\|$ is usually set to $\sqrt{\epsilon}$, where $\|A\|$ can be estimated through the computed eigenvalue distribution and ϵ is the relative machine precision.

Partial Reorthogonalization. The objective is to keep R_j orthogonal to all Q_i, $i < j$. The level of orthogonality among the blocks of vectors is then measured by $\eta_{i,j+1} = \| Q_i^T B Q_{j+1} \|$, and whenever this norm is greater than a given threshold, Q_{j+1} is orthogonalized against the corresponding Q_i.

It should be noted that the aforementioned norms are not explicitly computed, since they can be estimated and updated from the norms of A_j and B_j, applied to relation (6). A reasonable strategy is to reorthogonalize whenever $\tau_{j+1}^{(i)}$ or $\eta_{i,j+1}$ are greater than $\sqrt{\epsilon}\, np$. However, it is likely that the effectiveness of the selective orthogonalization or the partial reorthogonalization depends on the application and such strategies are sometimes used together [22]. Actually, in the present implementation, eigenvectors are computed only at

the end of a given run and selective orthogonalization will be performed only if a restart is required. Still, a modified partial reorthogonalization strategy is used [16, 22], in the sense that whenever $\eta_{i,j+1}$ reaches the threshold all the previous Q_j are taken into account. Therefore, instead of partial reorthogonalizations at possibly close steps, a full reorthogonalization is performed at some steps. Finally, for both orthogonalization schemes, both Q_{j+1} and Q_j are orthogonalized, since they are needed in the computation of Q_{j+2}. Such a strategy should assure a more adequate level of orthogonality among the subsequent blocks.

3 The Code Sections Examined

The basic approach for optimization of a given program is to detect the time-consuming parts and improve the corresponding code sections one by one. In this case, the following phases of restructuring could be applied [8]: *i*) use of the best set of compiler options, *ii*) modification of the sequential source code, and *iii*) improvement of the parallel code after it has been transformed by the compiler. It can be verified that the most important time consuming parts in the block Lanczos algorithm, in descending order, are

1. The factorization of A_σ and its repeated application to p vectors.
2. Orthogonalizations within and among blocks of Lanczos vectors, which involve operations b to e in Table 1, and also the partial reorthogonalization strategy.
3. Orthogonalizations among Ritz vectors and blocks of Lanczos vectors, which correspond to the selective orthogonalization strategy.
4. Evaluation of the reduced eigenproblem associated with the block tridiagonal matrix and the computation of Ritz vectors.

The p solutions using A_σ is not discussed here, since it is influenced by the pattern of the matrices and in many cases dominates the computational costs. Also, the cost for the solution of the reduced eigenproblem depends on p and on the number of steps, so that it varies independently of n and is almost negligible for very large applications. Therefore, the purpose is to speed up the second and third sets of operations listed above, as well as the computation of Ritz vectors.

The Lanczos algorithm is dominated by vector-scalar and vector-vector products, which can be expressed by matrix-vector and matrix-matrix products in the block case. Using the Basic Linear Algebra Subroutines (BLAS) notation, those operations are referred to as Level 1, 2 and 3, respectively. Then, one could employ a combination of _SCAL, _DOT, _AXPY, _GEMV, _GER and _GEMM to perform the computations in phases b to e, shown in Table 2. Such kernels are nowadays available in almost all computer scientific libraries and this helps in devising portable codes. Moreover, any restructuring could be performed in independent and specific code sections. On a multiprocessor machine, parallelization could be achieved inside the kernels or at loops with embedded kernels. However, load balance problems would occur for block sizes which are not multiples of the number of processors. On the other hand, for the kind of matrices involved in the block Lanczos basis generation

process, where $n \gg p$, it is not evident that a given level of BLAS would perform in the same way on different machines. In order to examine the impact of n, p and the BLAS level on different architectures, the following arrangements have been tested[§]:

phase b, orthogonalization of R_j against Q_{j-1} ($(p+1)pn$ flops):

level 1: _AXPY embedded in two loops (one for generating the column of R_j, and the other for generating the column of Q_{j-1} and the element of B_j^T);
level 2: _GEMV embedded in one loop (for generating the column of B_j^T);
level 3: no specific kernel available to profit from the pattern of B_j^T.

phase c, computation of A_j (p^2n flops):

level 1: _DOT embedded in two loops (one for generating the column of Q_j, and the other for generating the column of the array storing the product BR_j);
level 2: _GEMV embedded in one loop (for generating the column of the array storing the product BR_j);
level 3: _GEMM

phase d, orthogonalization of R_j against Q_j ($2p^2n$ flops):

level 1: _AXPY embedded in two loops (one for generating the column of R_j, and the other for generating the column of Q_j and the element of B_j^T);
level 2: _GEMV embedded in one loop (for generating the column of A_j);
level 3: _GEMM

phase e, factorization of R_j ($\frac{(5p+1)}{2}pn$ flops):

loop on all columns of R_j, _DOT and _SCAL to normalize one column each time and[¶]

level 1: _DOT and _AXPY embedded in one loop (for purging the normalized column from each column to its right);
level 2: _GEMV and _GER (for computing the orthogonalization factors and purging using an outer vector product);
level 3: not applicable, since this phase is basically a matrix-vector operation.

selective orthogonalization ($5pn$ flops):

level 1: _DOT and _AXPY embedded in one loop (for purging a Ritz vector from each column of R_j);
level 2: _GEMV and _GER (for computing the orthogonalization factors and purging using an outer vector product);
level 3: not applicable, since this phase is basically a matrix-vector operation.

[§]The partial reorthogonalization strategy is a combination of phases c and d, and the computation of Ritz vectors is similar to the operations in phase d.

[¶]This phase requires the product BR_j, which is updated in the same way as R_j. Therefore, R_j is transformed to Q_{j+1} and BR_j to BQ_{j+1} to be used in phase (a) of the next step.

Table 2: Simulation overview for $p = 6$ (level shown for best performance)

phase	Sun 10/41	IBM 950	Convex C220	Alliant FX/80	CRAY 2	CRAY C90
(b)	2	1	2	2	2	2
(c)	2 or 3	3	1	3	1	1
(d)	2 or 3	3	3	3	3	2 or 3
(e)	2	1	1	1	1	1
s.o.	1 or 2	2	1	2	1	1

The performance of the arrangements were examined on the super-scalar workstations SUN 10/41 and IBM Risc 6000/950, and shared memory multiprocessors Convex C220, Alliant FX/80, CRAY 2 and CRAY C90. For the CRAYs, n was taken between 30000 and 240000 (with increments of 30000), and for the others between 10000 and 100000 (with increments of 10000). The values of p were taken between 1 and 8, which are reasonable block sizes for current applications. Larger values usually require additional effort in the orthogonalizations and in the Gram-Schmidt factorization. It should be noted that the performances for $p = 1$ are not the best that can be obtained for the basic Lanczos algorithm, since even in this case the present implementation has a block structure. The matrices Q and R used in the experiments were generated with random numbers and the results are discussed in Appendix A. Anyhow, an overview of the results for the particular case $p = 6$ is given in Table 2, where s.o. indicates the selective orthogonalization.

4 Current Applications and Future Work

The global performance of the code, for all level implementations and best BLAS combinations, is currently being examined using several large problems, which are described in Table 3. The first problem belongs to the Harwell-Boeing matrix collection [7] and is associated with a finite element analysis of a skyscraper. It is interesting because the smallest eigenvalue (equal to zero) has multiplicity 118. The other problems are also related to finite element analyses and were kindly provided by D. Sorensen.

Another problem being examined comes from a molecular mechanics analysis, where A corresponds to the product $M^{-1/2}\nabla^2 E M^{-1/2}$, being E the potential energy, and M the matrix of the atomic masses (diagonal). Therefore, considering the eigenproblem (1), B is an identity matrix. However, a generalized eigenproblem formulation allows the consideration of the energy in terms of internal coordinates and the analysis of very large molecules. The characteristics of some cases are given in Table 4, where nz corresponds to the number of non zeros (upper triangle), and bw to the semibandwidth of the matrix. The dimension of each problem is three times the number of atoms. The first 16 eigenvalues of *arabinose*, for instance, are given in Table 5, as well as the related norms $\beta_j^{(i)}$ and the product $x^T Ax$ for each eigenvector. Such products have been computed only to verify the accuracy of the algorithm for this kind of application. The first 6 eigenvalues have no practical interest because they are associated with free (zero energy) molecular motions.

Table 3: Test cases characteristics

problem	n	characteristics
bcsstk25	15439	76 storey skyscraper
CT20	52329	auto engine model
NASAPWT	217918	wind tunel model
MN12	264002	automobile model

Table 4: Molecular dynamics applications

case	n	nz	bw
crambine	1188	134217	504
lysozyme	3795	490602	1137
arabinose	8592	1161360	1395

Table 5: Eigenvalues of arabinose

vector	λ	$x^T Ax$	$\beta_j^{(i)}$
1	-4.0968E-02	-4.0968E-02	1.1176E-15
2	-3.1209E-04	-3.1208E-04	2.9100E-15
3	-2.4069E-06	-2.3838E-06	4.2260E-15
4	-1.4992E-06	-1.4838E-06	2.5060E-15
5	1.1459E-07	1.2960E-07	3.4310E-15
6	1.5875E-06	1.6082E-06	3.7401E-15
7	2.0016E-06	2.0206E-06	1.5258E-15
8	6.6040E-06	6.6204E-06	2.8386E-15
9	4.0125E-04	4.0126E-04	3.4032E-15
10	8.1116E-04	8.1117E-04	1.8453E-15
11	1.1388E-03	1.1388E-03	1.6895E-15
12	1.4323E-03	1.4323E-03	5.5481E-21
13	1.5169E-03	1.5169E-03	1.8863E-15
14	2.2458E-03	2.2459E-03	6.0025E-15
15	2.9136E-03	2.9136E-03	1.0753E-10
16	3.2549E-03	3.2549E-03	1.0234E-09

The local experiments with all phases of the block Lanczos basis generation process indicated that different performances are obtained with the BLAS levels, depending on n and p. Therefore, the next objective is to choose automatically the most appropriate BLAS level for each operation on a given computer at run time. It should be noted that the operations are nested in a loop and any performance improvement would be multiplied by the number of Lanczos steps. However, on supercomputers like the CRAYs a significant variation is found for some combinations of n and p and additional experiments (using fixed leading array dimensions, for example) need to be performed in order to obtain a more regular behaviour.

On the other hand, the generation of large bases (long runs) requires a large storage space for the vectors (or more I/O operations if the vectors do not fit into fast access memory) and more partial reorthogonalizations or selective orthogonalizations. Therefore, it appears that more efficient ways of restarting and spectrum slicing should be also investigated, based on the information that the algorithm computes in each run.

Another very important point to be studied is the factorization of A_σ and its repeated application to p vectors. For specific problems, different decomposition strategies could be possibly applied, and even an iterative solver, although the inertia information would not be available.

Acknowledgment. The author would like to thank D. Sorensen, Rice University, Houston, USA, for having provided large test problems, and Yves-Henri Sanejouand, IRSAMC, Paul Sabatier University, Toulouse, France, for having provided eigenproblems from molecular mechanics analyses.

References

[1] A. Björck. Numerics of Gram-Schmidt Orthogonalization. *Linear Algebra and Its Applications*, 197,198:297–316, 1994.

[2] S. W. Bostic and R. E. Fulton. Implementation of the Lanczos Method for Structural Vibration Analysis on a Parallel Computer. *Computers & Structures*, 25:395–403, 1987.

[3] T. Braconnier. The Arnoldi-Tchebycheff Algorithm for Solving Large Symmetric Eigenproblems. Technical Report TR/PA/93/25, CERFACS, Toulouse, France, 1993.

[4] F. Chatelin. *Valeurs Propres de Matrices*. Masson, Paris, France, 1988.

[5] A. T. Chronopoulos and C. W. Gear. s-step Iterative Methods for Symmetric Linear Systems. *J. Comp. and Applied Math.*, 25:153–168, 1989.

[6] J. W. Daniel, W. B. Cragg, L. Kaufman, and G. H. Stewart. Reorthogonalization and Stable Algorithms for Updating the Gram-Schmidt QR Factorization. *Math. of Comp.*, 30:772–795, 1976.

[7] I. S. Duff, R. G. Grimes, and J. G. Lewis. User's Guide for the Harwell-Boeing Sparse Matrix Collection (Release I). Technical Report TR/PA/92/86, CERFACS, Toulouse, France, 1992.

[8] R. Eigenmann. Towards a Methodology of Optimizing Programs for High Performance Computers. Technical Report 1178, CSRD, University of Illinois at Urbana-Champaign, Urbana-Champaign, USA, 1992.

[9] T. Ericsson and A. Ruhe. The Spectral Transformation Lanczos Method for the Numerical Solution of Large Sparse Generalized Symmetric Eigenvalue Problems. *Mathematics of Computation*, 35:1251–1268, 1980.

[10] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An Implementation of the Look-Ahead Lanczos Algorithm for Non-Hermitian Matrices, Part I. Technical Report 90.45, RIACS, NASA Ames Research Center, Columbia, USA, 1990.

[11] R. W. Freund, M. H. Gutknecht, and N. M. Natchigal. An Implementation of the Look-Ahead Lanczos Algorithm for Non-Hermitian Matrices. *SIAM J. Sci. Comput.*, 14:137–158, 1993.

[12] R. W. Freund and N. M. Nachtigal. An Implementation of the Look-Ahead Lanczos Algorithm for Non-Hermitian Matrices, Part II. Technical Report 90.46, RIACS, NASA Ames Research Center, Columbia, USA, 1990.

[13] S. Godet-Thobie. Valeurs Propres de Matrices Fortement Non Normales en Grande Dimension. Technical Report TH/PA/93/06, CERFACS, Toulouse, France, 1993.

[14] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, USA, third edition, 1996.

[15] G. H. Golub and R. Underwood. The Block Lanczos Method for Computing Eigenvalues. In J. R. Rice, editor, *Mathematical Sotware III*, pages 361–377. Academic Press, Inc., 1977.

[16] R. G. Grimes, J. G. Lewis, and H. D. Simon. A Shifted Block Lanczos Algorithm for Solving Sparse Symmetric Eigenvalue Problems. Technical Report RNR-91-012, Boeing Computer Services, Seattle, USA, 1991.

[17] M. T. Jones and M. L. Patrick. The Lanczos Algorithm for the Generalized Symmetric Eigenproblem on Shared-Memory Architectures. Technical Report MCS-P182-0990, Argonne National Laboratory, Argonne, USA, 1990.

[18] M. T. Jones and M. L. Patrick. The Use of Lanczos's Method to Solve the Large Generalized Symmetric Eigenvalue Problem in Parallel. Technical Report 90-48, ICASE, Hampton, USA, 1990.

[19] S. Kim and A. T. Chronopoulos. A Class of Lanczos-Like Algorithms Implemented on Parallel Computers. Technical Report TR 89-49, Computer Science Dept., University of Minnesota, Minneapolis, USA, 1989.

[20] S. Kim and A. T. Chronopoulos. s-step Lanczos and Arnoldi Methods on Parallel Computers. Technical Report TR 89-83, Computer Science Dept., University of Minnesota, Minneapolis, USA, 1989.

[21] J. Malard. *Block Solvers for Dense Linear Systems on Local Memory Multiprocessors*. PhD thesis, School of Computer Science, McGill University, Montreal, Canada, 1992.

[22] B. Nour-Omid. The Lanczos Algorithm for Solution of Large Generalized Eigenproblem. In T. J. R. Hughes, editor, *The Finite Element Method*, pages 582–630, Englewood Cliffs, USA, 1987. Prentice Hall International Editions.

[23] B. Nour-Omid, B. N. Parlett, T. Ericsson, and P. S. Jensen. How to Implement the Spectral Transformation. *Mathematics of Computation*, 48:663–673, 1987.

[24] B. N. Parlett. Notes on Lanczos Algorithms for a SIAM Short Course on Large-Scale and Parallel Matrix Computations in Control, Systems and Signal Processing, November 1990.

[25] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM (Classics in Applied Mathematics), Philadelphia, USA, 1998.

[26] B. N. Parlett, B. Nour-Omid, and J. Natvig. Implementation of Lanczos Algorithms on Vectors Computers. In R. W. Numrich, editor, *Supercomputers Applications*, pages 1–17. Plenum Press, 1985.

[27] B. N. Parlett and D. S. Scott. The Lanczos Algorithm with Selective Orthogonalization. *Mathematics of Computation*, 33:217–238, 1979.

[28] C. Rajakumar and C. R. Rogers. The Lanczos Algorithm Applied to Unsymmetric Generalized Eigenvalue Problem. *Int. J. for Numer. Meth. in Eng.*, 32:1009–1026, 1991.

[29] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, Manchester, England, 1992.

[30] H. D. Simon. The Lanczos Algorithm with Partial Reorthogonalization. *Mathematics of Computation*, 42:115–142, 1984.

[31] D. C. Sorensen. Implicit Application of Polynomial Filters in a k-step Arnoldi Method. *SIAM J. Matrix Anal. Appl.*, 13:357–385, 1992.

[32] Q. Ye. A Convergence Analysis for Nonsymmetric Lanczos Algorithms. *Mathematics of Computation*, 56:677–691, 1991.

A Simulation Details

This appendix shows some instances of the computational performances for the arrangements described in Section 3. The performance (*time*) is measured using different values of n and is expressed in seconds. A dashed line indicates results for Level 1, a dash-dotted line for Level 2, and a solid line for Level 3 BLAS arrangements. Most of the pictures correspond to $p = 6$, in order to provide a general view. If $p = 1$ the Level 1 is recommended for all cases, since it has been verified that the Levels 2 and 3 can lead to very poor performances. Machine specific comments are given in the following paragraphs.

Sun 10/41. The computational performances on the Sun 10/41 are given in terms of user CPU time. It should be noted that no specific BLAS library is available on the Sun 10/41. Therefore, the library employed was generated with the BLAS source code using only the -O optimization option provided by the compiler. An important oscillation on the performance has been noticed for $p < 3$ in all computational phases, probably related to data management and the SUN clock resolution. The Level 1 implementation is therefore recommended in such cases. An improvement can be obtained with the Level 2 implementation for the operations in phases b and e (but in a smaller proportion), as shown in Figs. 1 and 4. However, the situation is also typical for other values of the block size. On the other hand, the performances of the Levels 2 and 3 implementations are almost the same for phases c and d, as indicated in Figs. 2 and 3. For those phases, the asterisk represents the performance of an experimental Level 3 tuned BLAS implementation. As can be seen, a significant improvement can be obtained in phase c (Fig. 2), which encourages the development of the tuned version to deal with odd values of p in phase d (Fig. 3). Concerning the selective orthogonalization, the performances are fundamentally the same for Levels 1 and 2, and are not plotted.

IBM Risc 6000/950. The computational performances on the IBM Risc 6000/950 are given in terms of user plus system CPU time. Similarly to the Sun 10/41, an important oscillation has been noticed for $p < 3$ in all computational phases, and the Level 1 implementation is thus recommended in such cases. The performances of the Level 1 implementations are generally better in phases b and e, as shown in Figs. 5 and 8, and also for the selective orthogonalization. However, as the block size increases ($p > 6$), the performance of the Level 2 was noticed to be very similar to that of the Level 1, and sometimes even better for the selective orthogonalization. On the other hand, a significant improvement is obtained with the Level 3 implementations in phases c and d, as shown in Figs. 6 and 7. The situation is typical for other values of the block size.

Convex C220. The computational performances on the Convex C220 (with one processor) are given in terms of user CPU time. The vector architecture and the Convex scientific library favour the Level 1 implementations for phases c, e and the selective orthogonalization, as shown in Figs. 10, 12 and 13. On the other hand, the Level 2 implementation leads to a significant improvement in phase b, as can be seen in Fig. 9. It should be noted that

even in phase d, where the Level 3 implementation is generally more efficient, the Level 2 also performs well, as shown in Fig. 11. This behaviour stands for other values of p.

Alliant FX/80. The computational performances on the Alliant FX/80 are given in terms of user CPU time, using 4 and 8 processors. On this machine, the higher level implementations improve the computational performances for phases b, c and d, as can be seen in Figs. 14, 15 and 16. On the other hand, the Level 1 is more efficient in phase e, as shown in Fig. 17. Concerning the selective orthogonalization, the Level 1 is more efficient for $p < 6$, otherwise the Level 2 should be used. It has been noticed that the use of a different number of processors can also influence the performances. A very good speed-up is obtained with the Level 1 implementation, as can be verified in Fig. 15, for example, in spite of the better performance obtained with the Level 3. It seems that larger values of n and p are required to obtain similar speed-up's with the higher level implementations.

CRAY 2. The computational performances on the CRAY 2 are given in terms of real-time clock values, for 2 and 4 processors, running in a stand alone mode (the benchmark queue). The vector architecture and the library available on the CRAY 2 favour the Level 1 implementation in phases c, e and selective orthogonalization, as indicated in Figs. 19, 21 and 22. On the other hand, the higher level implementations are more appropriate for the computations in phases b and d, as shown in Figs. 18 and 20. The peaks in some plots probably indicate that the corresponding implementantions are sensitive to data management. In addition, no speed-up has been obtained for all implementations and phases, which means that specific tuned kernels should be developed for the CRAY 2.

CRAY C90. The computational performances on the CRAY C90 are given in terms of real-time clock values, for 2 and 4 processors, running in a stand alone mode (the benchmark queue). Similarly to the CRAY 2, the vector architecture and the library available favour the Level 1 implementation in phases c, e and selective orthogonalization, as depicted in Figs. 24 and 26 and 27. On the other hand, the higher level implementations are adequate for the computations in phases b and d, as shown in Figs. 23 and 25. However, it has been verified that the performances switch between the Levels 2 and 3 depending on the block size an on the number of processors, but a very good speed-up has been obtained in some cases, as can be seen for example in Fig. 24.

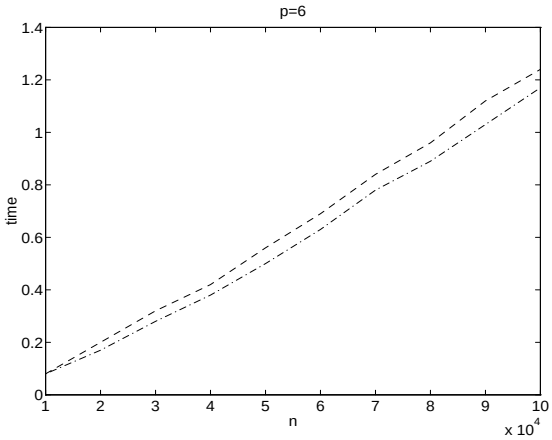

Figure 1: SUN 10/41, orthog. of R_j against Q_{j-1}

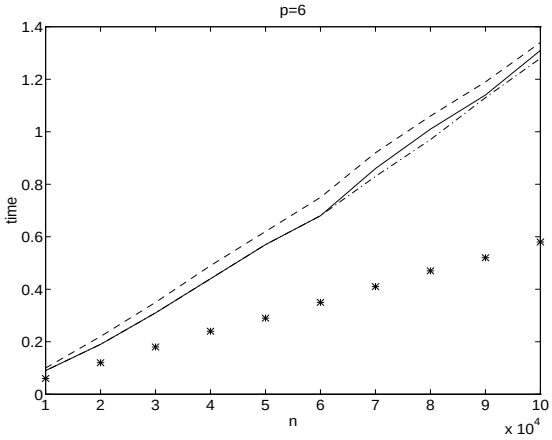

Figure 2: SUN 10/41, comp. of A_j

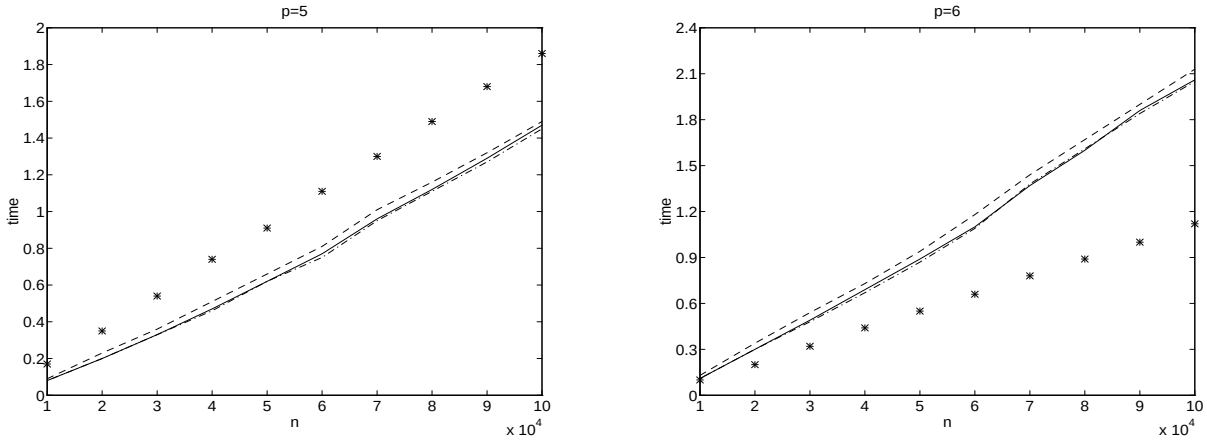

Figure 3: SUN 10/41, orthog. of R_j against Q_j

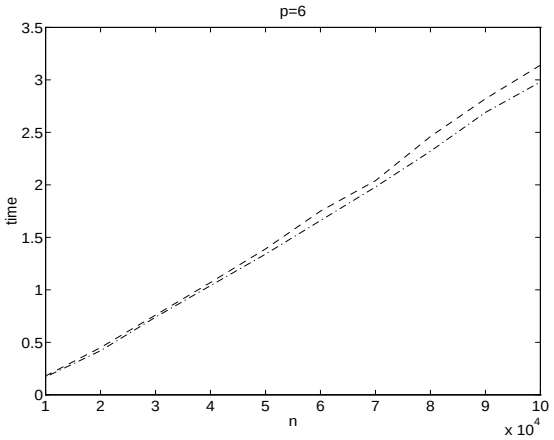

Figure 4: SUN 10/41, factor. of R_j

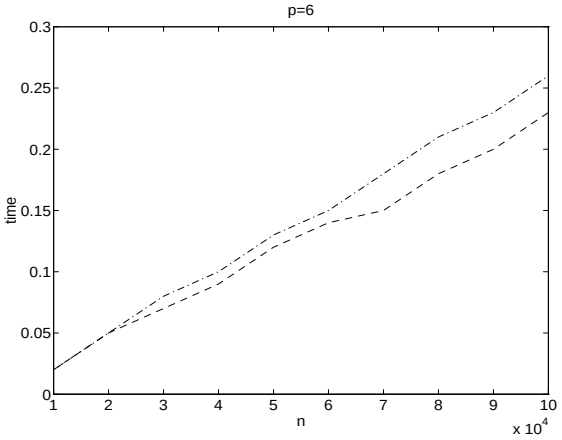

Figure 5: IBM Risc 6000/950, orthog. of R_j against Q_{j-1}

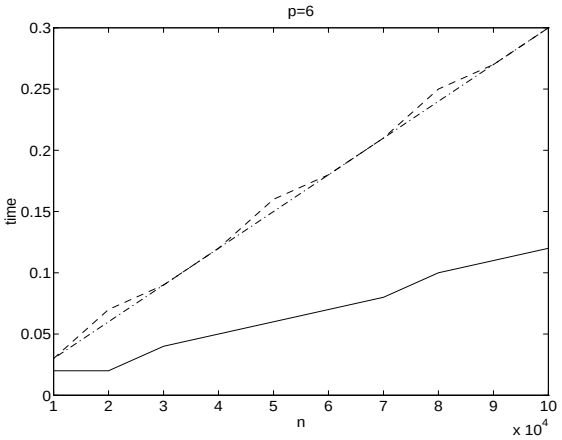

Figure 6: IBM Risc 6000/950, comp. of A_j

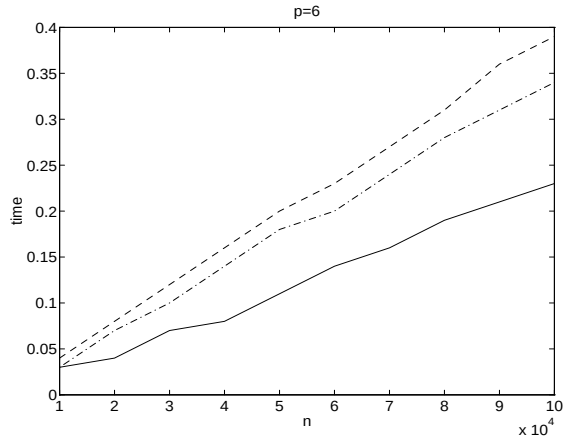

Figure 7: IBM Risc 6000/950, orthog. of R_j against Q_j

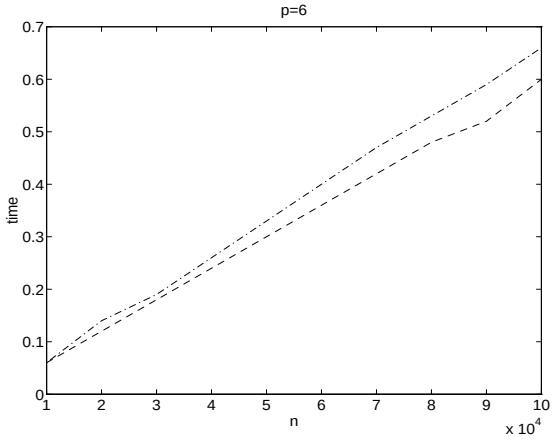

Figure 8: IBM Risc 6000/950, factor. of R_j

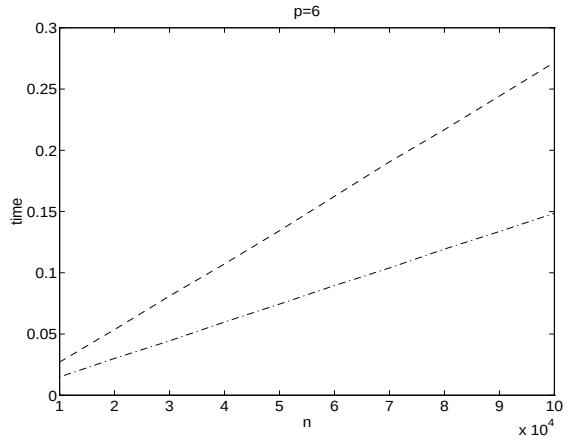

Figure 9: Convex C220 (1 proc.), orthog. of R_j against Q_{j-1}

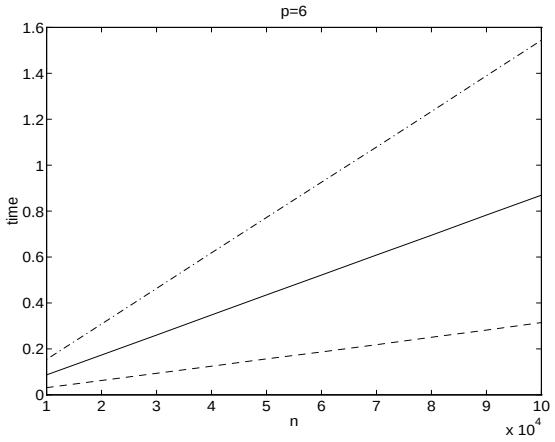

Figure 10: Convex C220 (1 proc.), comp. of A_j

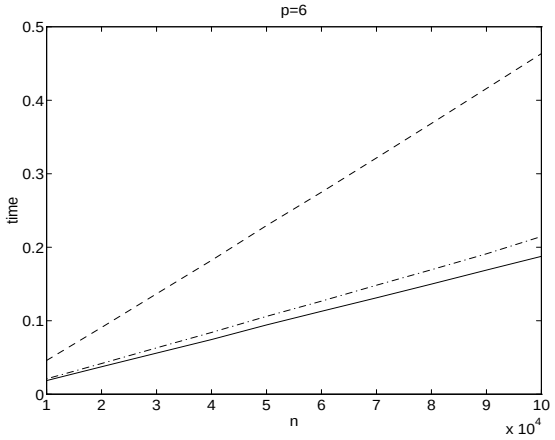

Figure 11: Convex C220 (1 proc.), orthog. of R_j against Q_j

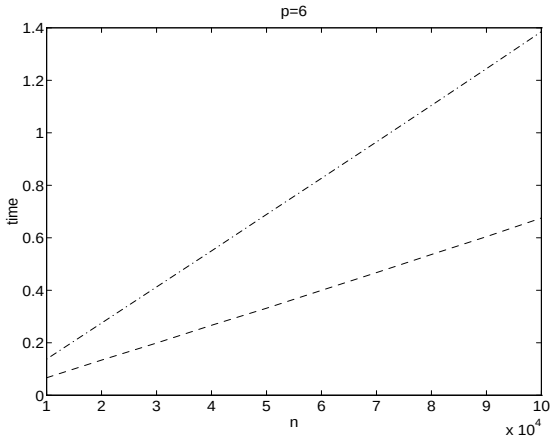

Figure 12: Convex C220 (1 proc.), factor. of R_j

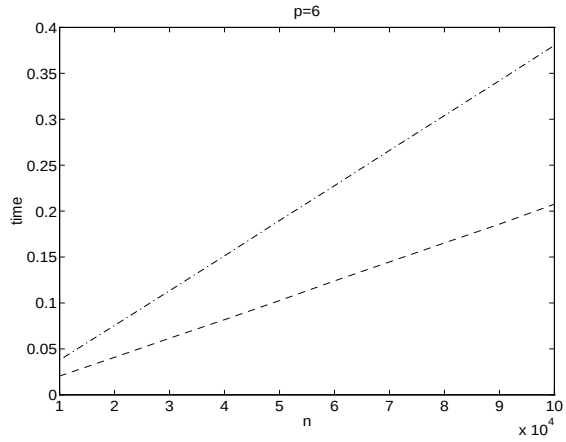

Figure 13: Convex C220 (1 proc.), selective orthogonalization

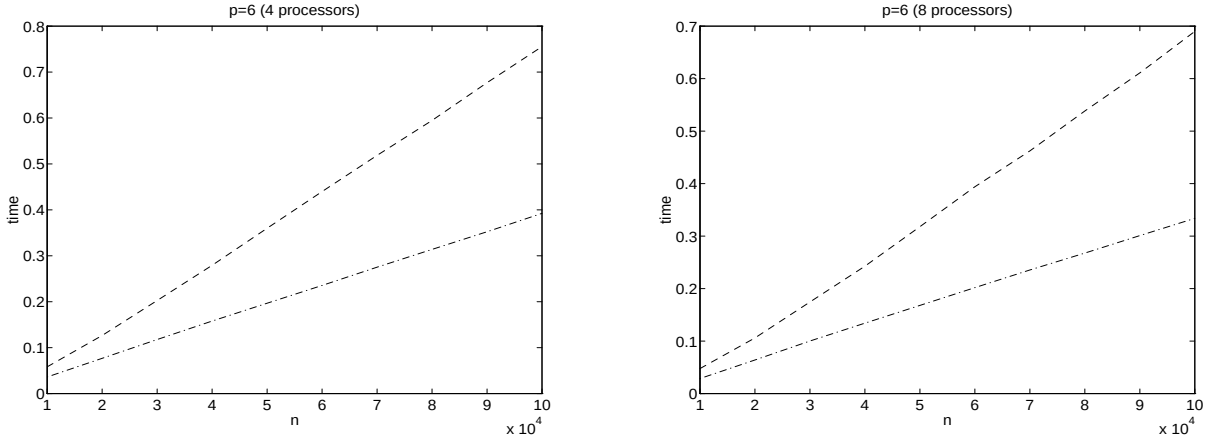

Figure 14: Alliant (4 and 8 proc.), orthog. of R_j against Q_{j-1}

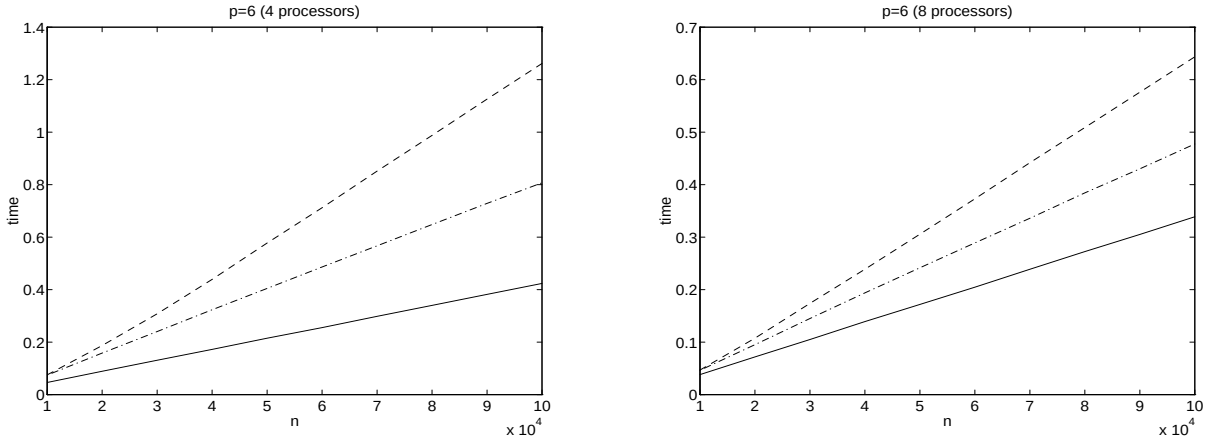

Figure 15: Alliant (4 and 8 proc.), comp. of A_j

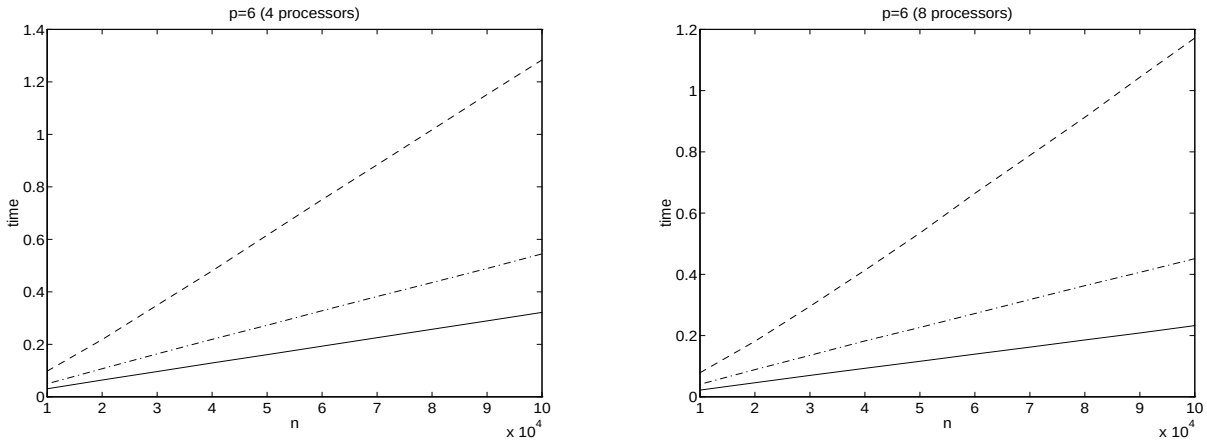

Figure 16: Alliant (4 and 8 proc.), orthog. of R_j against Q_j

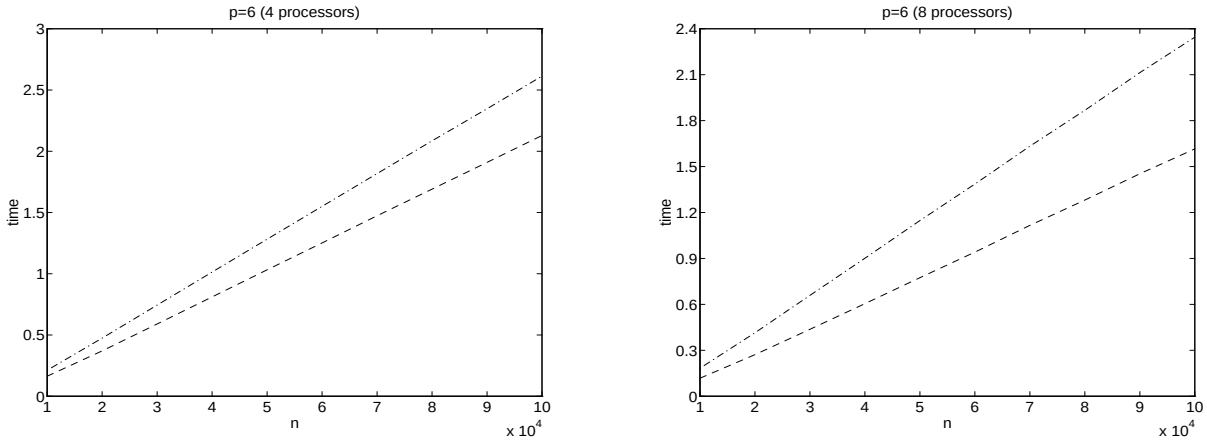

Figure 17: Alliant (4 and 8 proc.), factor. of R_j

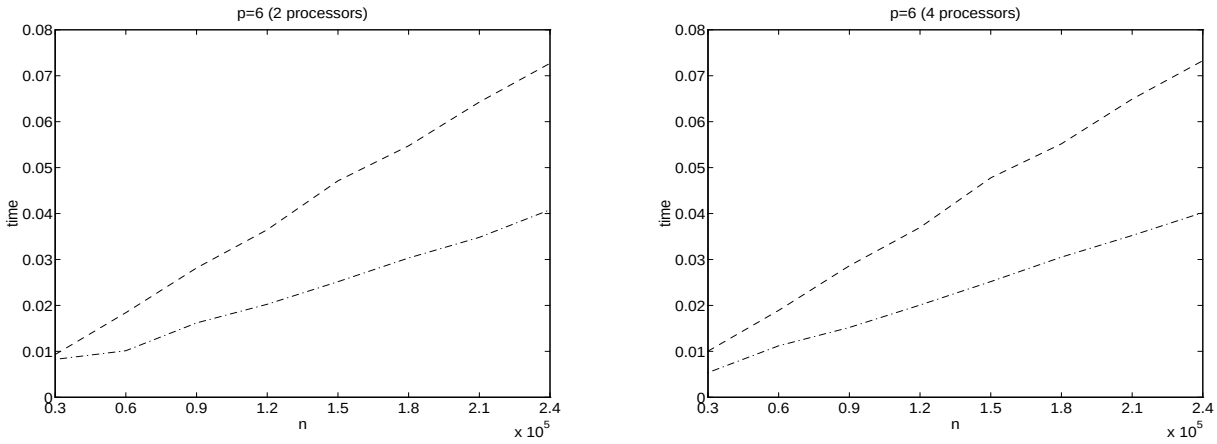

Figure 18: CRAY 2 (2 and 4 proc.), orthog. of R_j against Q_{j-1}

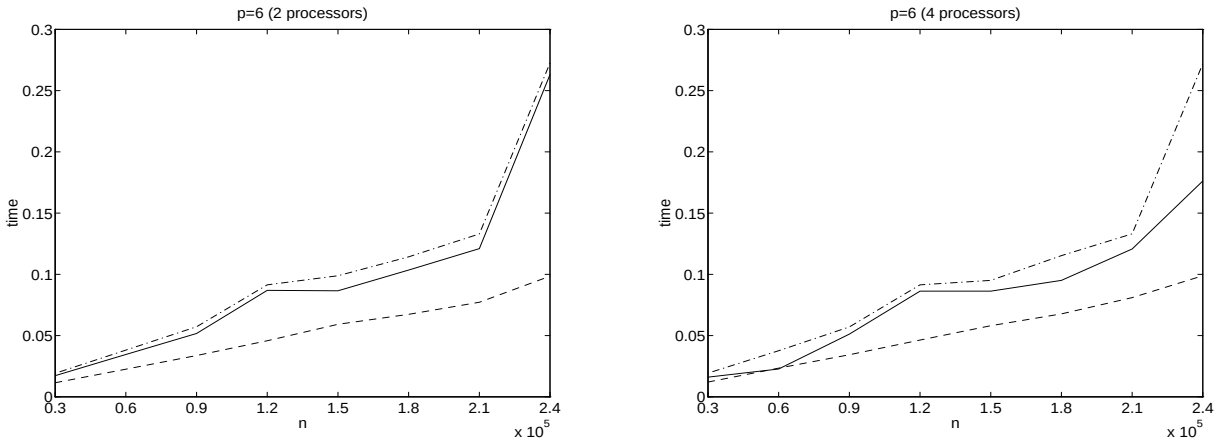

Figure 19: CRAY 2 (2 and 4 proc.), comp. of A_j

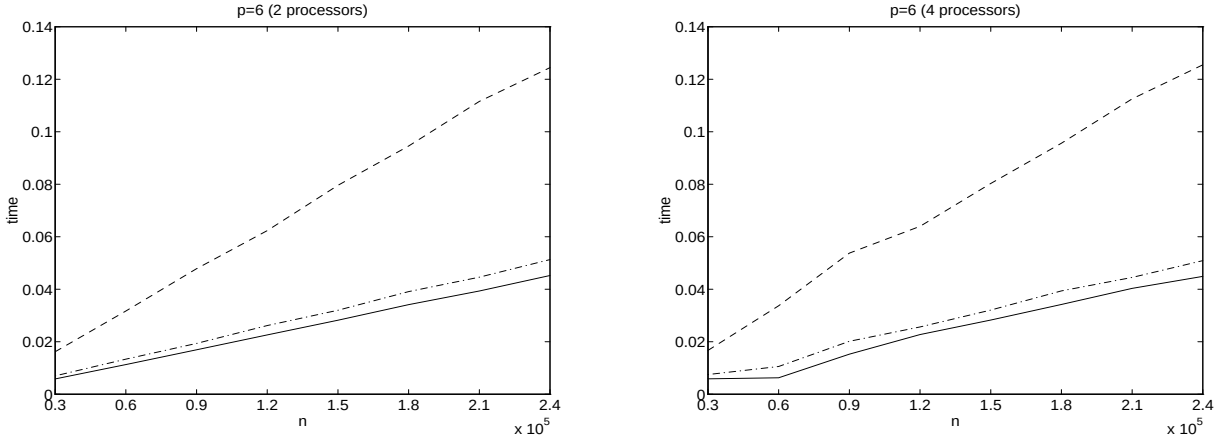

Figure 20: CRAY 2 (2 and 4 proc.), orthog. of R_j against Q_j

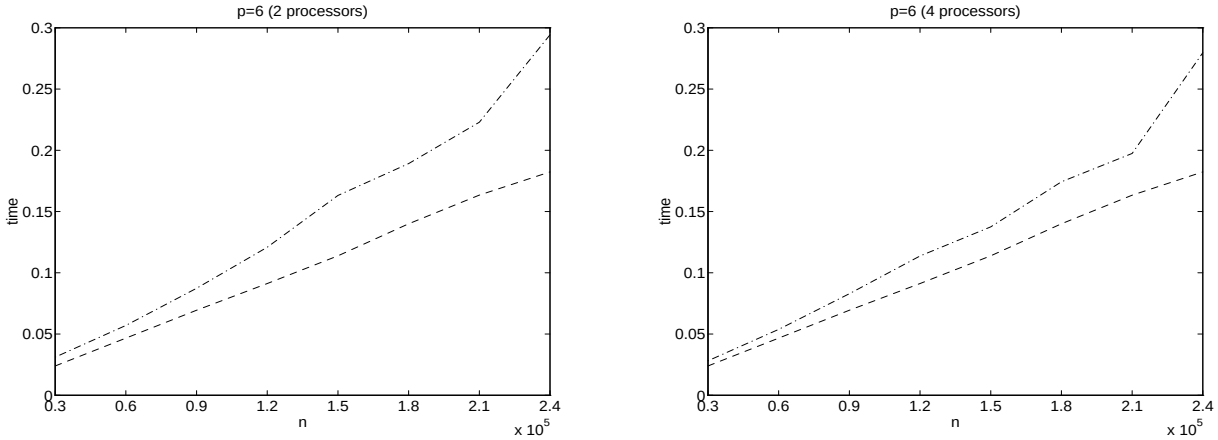

Figure 21: CRAY 2 (2 and 4 proc.), factor. of R_j

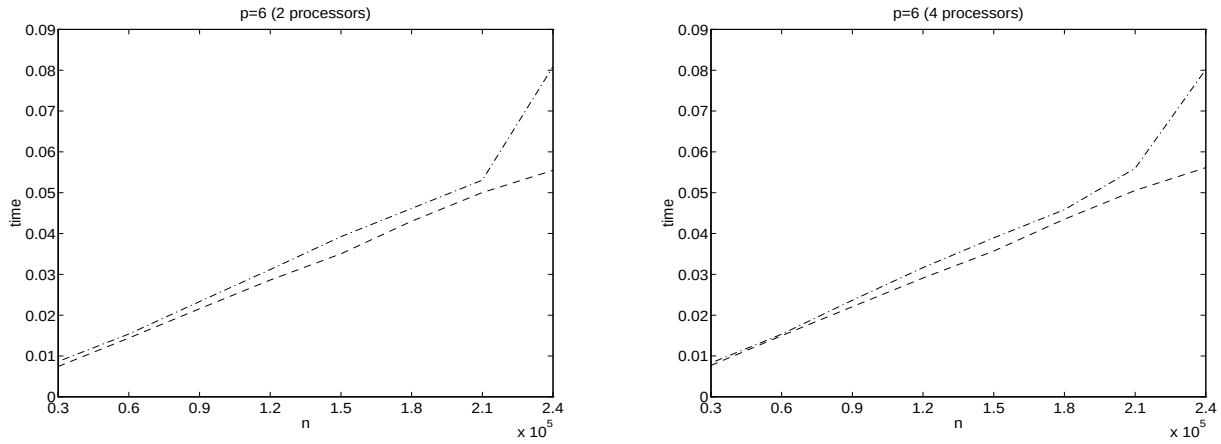

Figure 22: CRAY 2 (2 and 4 proc.), selective orthogonalization

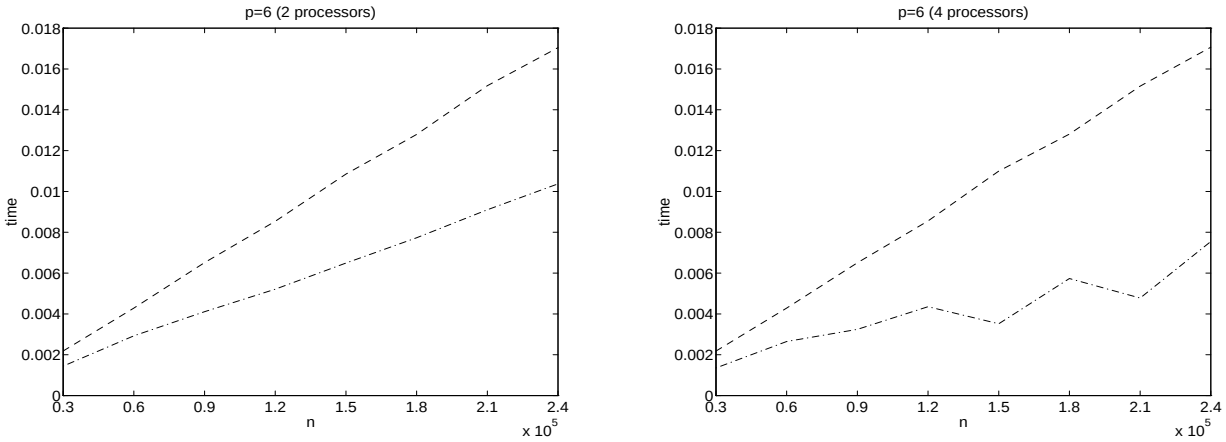

Figure 23: CRAY C90 (2 and 4 proc.), orthog. of R_j against Q_{j-1}

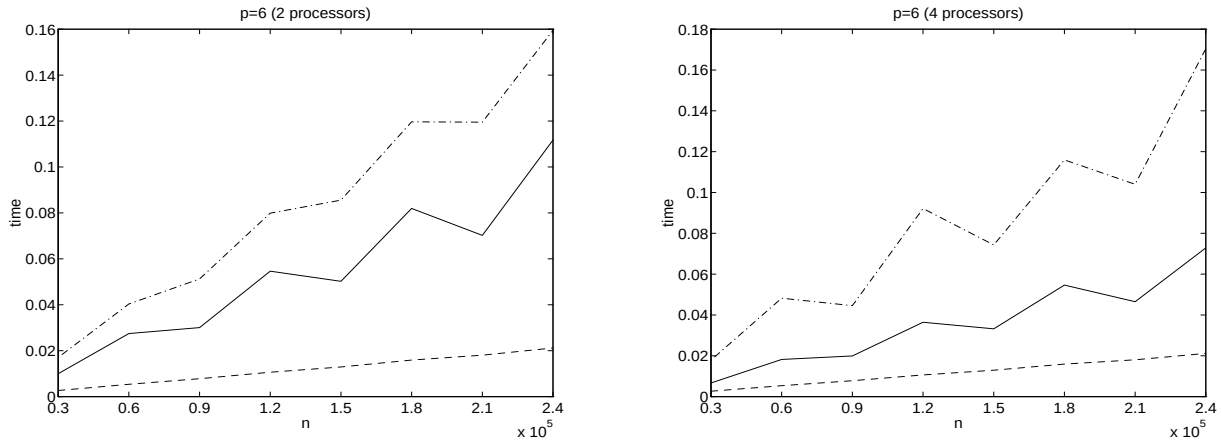

Figure 24: CRAY C90 (2 and 4 proc.), comp. of A_j

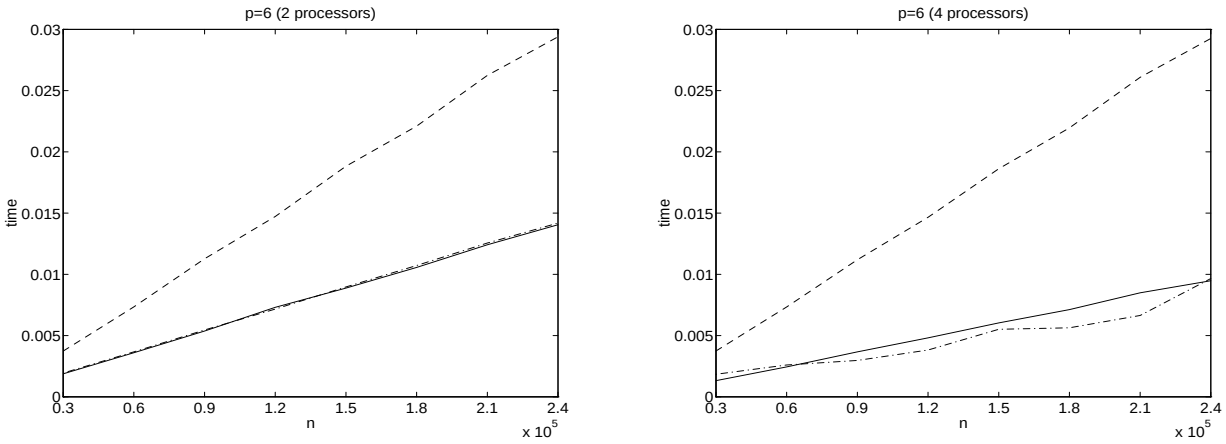

Figure 25: CRAY C90 (2 and 4 proc.), orthog. of R_j against Q_j

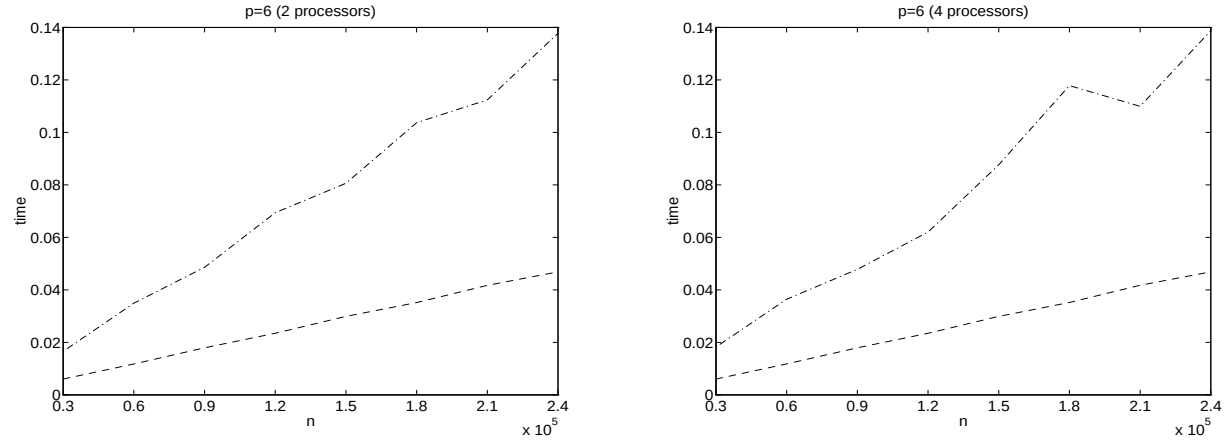

Figure 26: CRAY C90 (2 and 4 proc.), factor. of R_j

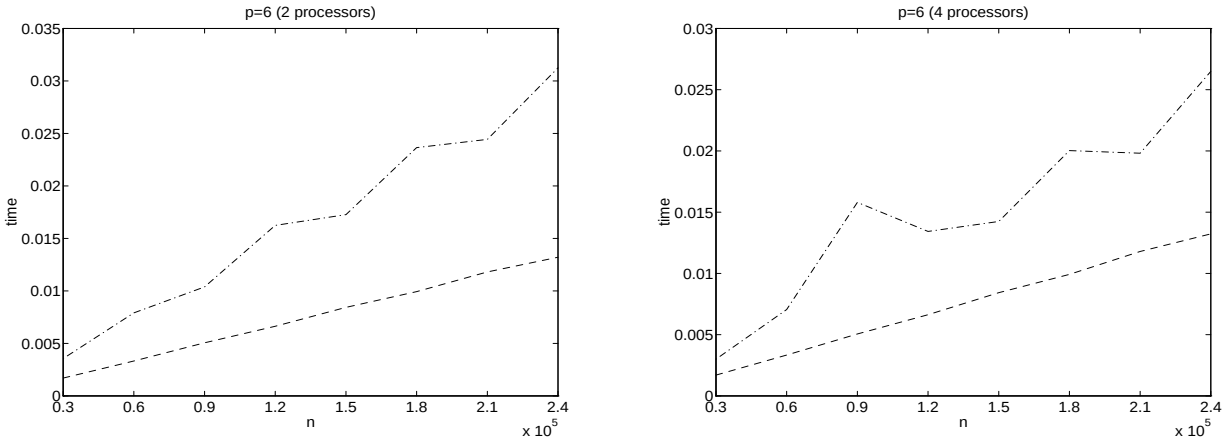

Figure 27: CRAY C90 (2 and 4 proc.), selective orthogonalization